HotSpot Control

See Also Example Properties Methods Events

Specifies a region for exposing events.

Remarks

Assigning actions to the **HotSpot** control's **MouseEnter** and **MouseExit** events determines what happens when a user moves the mouse pointer over the **HotSpot**.

You can also use **HotSpot** as an alternative to image maps in HTML. To do this, place multiple **HotSpot** controls over an **Image** control and assign actions to the **HotSpot** controls' events. By default, the **HotSpot** is invisible at run time because the default value of the **BackStyle** property is **Transparent** and the **BorderStyle** property is **None**.

To make your HTML Layout more accessible to keyboard-only users, assign actions to the **HotSpot** control's **Enter** event and make sure that the **Enabled** property is set to **True**. At run time, keyboard-only users will then be able to tab to the **HotSpot** and press ENTER to trigger the event.

The default event for a **HotSpot** is the Click event.

HTML Layout Control

References an HTML Layout and renders it at run time.

Remarks

An HTML Layout is a WYSIWYG drawing board to which you can add multiple controls. You can draw controls in the precise sizes and locations you want, group and align them, and even put one control on top of another.

The ActiveX Control Pad saves each HTML Layout in a file format with an .alx extension. When you insert an HTML Layout into HTML, the ActiveX Control Pad adds an **HTML Layout** control for each layout that you insert. The **HTML Layout** control is what actually renders the HTML Layout at run time.

Image Control

See Also Example Properties Methods Events

Displays a picture.

Remarks

The **Image** control lets you crop, size, or zoom a picture, but does not allow you to edit the contents of the picture. For example, you can't use **Image** to change the colors in the picture, to make the picture transparent, or to refine the image of the picture. You must use image editing software for these purposes.

Image supports the following formats:

- .gif ('87 and '89)
- .jpg
- .wmf
- .bmp

Note The picture is not actually embedded into the control. The control references the picture at run time based on the URL specified by the **PicturePath** property

The default event for **Image** is the Click event.

BeforeDragOver Event

See Also Example Applies To

Occurs when a drag-and-drop operation is in progress.

Syntax

For TabStrip

Private Sub *object*_**BeforeDragOver(** *index* **As Long**, **ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *DragState* **As fmDragState**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**

For other controls

Private Sub *object*_**BeforeDragOver(ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *DragState* **As fmDragState**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**

The **BeforeDragOver** event syntax has these parts:

Part	Description
object	Required. A valid object name.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle the event.
ctrl	Required. The control being dragged over.
Data	Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a **DataObject**.
X, Y	Required. The horizontal and vertical coordinates of the control's position. Both coordinates are measured in points. *X* is measured from the left edge of the control; *Y* is measured from the top of the control.
DragState	Required. Transition state of the data being dragged.
X, Y	Required. The horizontal and vertical coordinates of the control's position. Both coordinates are measured in points. *X* is measured from the left edge of the control; *Y* is measured from the top of the control..
Effect	Required. Operations supported by the drop source.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.

Settings

The settings for *DragState* are:

Constant	Value	Description
fmDragStateEnter	0	Mouse pointer is within range of a target.
fmDragStateLeave	1	Mouse pointer is outside the range of a target.
fmDragStateOver	2	Mouse pointer is at a new position, but remains within range of the same target.

The settings for *Effect* are:

Constant	Value	Description
fmDropEffectNone	0	Does not copy or move the drop source to the drop target.

fmDropEffectCopy	1	Copies the drop source to the drop target.
fmDropEffectMove	2	Moves the drop source to the drop target.
fmDropEffectCopyOrMove	3	Copies or moves the drop source to the drop target.

The settings for *Shift* are:

Constant	**Value**	**Description**
fmShiftMask	1	SHIFT was pressed.
fmCtrlMask	2	CTRL was pressed.
fmAltMask	4	ALT was pressed.

Remarks

Use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. When a drag-and-drop operation is in progress, the system initiates this event when the user moves the mouse, or presses or releases the mouse buttons. The mouse pointer position determines the target object that receives this event. You can determine the state of the mouse pointer by examining the *DragState* argument.

When a control handles this event, you can use the *Effect* argument to identify the drag-and-drop action to perform. When *Effect* is set to **fmDropEffectCopyOrMove**, the drop source supports a copy (**fmDropEffectCopy**), move (**fmDropEffectMove**), or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectCopy**, the drop source supports a copy or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectMove**, the drop source supports a move or a cancel (**fmDropEffectNone**) operation.

When *Effect* is set to **fmDropEffectNone**. the drop source supports a cancel operation.

Most controls do not support drag-and-drop while *Cancel* is **False**, which is the default setting. This means the control rejects attempts to drag or drop anything on the control, and the control does not initiate the BeforeDropOrPaste event. The **TextBox** and **ComboBox** controls are exceptions to this; these controls support drag-and-drop operations even when *Cancel* is **False**.

BeforeDropOrPaste Event

See Also Example Applies To

Occurs when the user is about to drop or paste data onto an object.

Syntax

For TabStrip
Private Sub *object*_**BeforeDropOrPaste(** *index* **As Long**, **ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Action* **As fmAction**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**
For other controls
Private Sub *object*_**BeforeDropOrPaste(ByVal** *Cancel* **As MSForms.ReturnBoolean**, **ByVal** *Action* **As fmAction**, **ByVal** *Data* **As DataObject**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single**, **ByVal** *Effect* **As MSForms.ReturnEffect**, **ByVal** *Shift* **As fmShiftState)**

The **BeforeDropOrPaste** event syntax has these parts:

Part	Description
object	Required. A valid object name.
index	Required. The index of the control that the drop or paste operation will affect.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle the event.
ctrl	Required. The target control.
Action	Required. Indicates the result, based on the current keyboard settings, of the pending drag-and-drop operation.
Data	Required. Data that is dragged in a drag-and-drop operation. The data is packaged in a **DataObject**.
X, Y	Required. The horizontal and vertical position of the mouse pointer when the drop occurs. Both coordinates are measured in points. *X* is measured from the left edge of the control; *Y* is measured from the top of the control..
Effect	Required. Effect of the drag-and-drop operation on the target control.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.

Settings

The settings for *Action* are:

Constant	Value	Description
fmActionPaste	2	Pastes the selected object into the drop target.
fmActionDragDrop	3	Indicates the user has dragged the object from its source to the drop target and dropped it on the drop target.

The settings for *Effect* are:

Constant	Value	Description
fmDropEffectNone	0	Does not copy or move the drop source to the drop target.

fmDropEffectCopy	1	Copies the drop source to the drop target.
fmDropEffectMove	2	Moves the drop source to the drop target.
fmDropEffectCopyOrMove	3	Copies or moves the drop source to the drop target.

The settings for *Shift* are:

Constant	**Value**	**Description**
fmShiftMask	1	SHIFT was pressed.
fmCtrlMask	2	CTRL was pressed.
fmAltMask	4	ALT was pressed.

Remarks

For a **TabStrip**, VBScript initiates this event when it transfers a data object to the control.

For other controls, the system initiates this event immediately prior to the drop or paste operation.

When a control handles this event, you can update the *Action* argument to identify the drag-and-drop action to perform. When *Effect* is set to **fmDropEffectCopyOrMove**, you can assign *Action* to **fmDropEffectNone**, **fmDropEffectCopy**, or **fmDropEffectMove**. When *Effect* is set to **fmDropEffectCopy** or **fmDropEffectMove**, you can reassign *Action* to **fmDropEffectNone**. You cannot reassign *Action* when *Effect* is set to **fmDropEffectNone**.

Click Event

See Also Example Applies To

Occurs in one of two cases:

- The user clicks a control with the mouse.
- The user selects a specific value for a control with more than one possible value.

Syntax

For all controls
  Private Sub *object*_**Click( )**

The **Click** event syntax has these parts:

Part	Description
object	Required. A valid object.

Remarks

Of the two cases where the Click event occurs, the first case applies to the **CommandButton**, **Image**, **Label**, **ScrollBar**, and **SpinButton**. The second case applies to the **CheckBox**, **ComboBox**, **ListBox**, **TabStrip**, **TextBox**, and **ToggleButton**.

The following are examples of actions that initiate the Click event:

- Clicking a blank area of an HTML Layout or a disabled control (other than a list box) on the HTML Layout.
- Clicking a **CommandButton**. If the command button doesn't already have the focus, the Enter event occurs before the Click event.
- Pressing the SPACEBAR when a **CommandButton** has the focus.
- Clicking a control with the left mouse button (left-clicking).
- Pressing a control's accelerator key.

When the Click event results from clicking a control, the sequence of events leading to the Click event is:

1. MouseDown
2. MouseUp
3. Click

For some controls, the Click event occurs when the **Value** property changes. However, using the Change event is the preferred technique for detecting a new value for a property. The following are examples of actions that initiate the Click event due to assigning a new value to a control:

- Clicking a **CheckBox** or **ToggleButton**, pressing the SPACEBAR when one of these controls has the focus, pressing the accelerator key for one of these controls, or changing the value of the control in code.
- Changing the value of an **OptionButton** to **True**. Setting one **OptionButton** in a group to **True** sets all other buttons in the group to **False**, but the Click event occurs only for the button whose value changes to **True**.
- Selecting a value for a **ComboBox** or **ListBox** so that it unquestionably matches an item in the control’s drop-down list. For example, if a list is not sorted, the first match for characters typed in the edit region may not be the only match in the list, so choosing such a value does not initiate the Click event. In a sorted list, you can use entry-matching to ensure that a selected value is a unique match for text the user types.

The Click event is not initiated when **Value** is set to **Null**.

Note Left-clicking changes the value of a control, thus it initiates the Click event. Right-clicking does not change the value of the control, so it does not initiate the Click event.

DblClick Event

See Also Example Applies To

Occurs when the user points to an object and then clicks a mouse button twice.

Syntax

For TabStrip
Private Sub *object*_**DblClick(** *index* **As Long**, *Cancel* **As MSForms.ReturnBoolean)**
For other controls
Private Sub *object*_**DblClick(** *Cancel* **As MSForms.ReturnBoolean)**

The **DblClick** event syntax has these parts:

Part	Description
object	Required. A valid object.
index	Required. The position of a **Tab** object within a **Tabs** collection.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle the event.

Remarks

For this event to occur, the two clicks must occur within the time span specified by the system's double-click speed setting.

For controls that support Click, the following sequence of events leads to the DblClick event:

1. MouseDown
2. MouseUp
3. Click
4. DblClick

If a control, such as **TextBox**, does not support Click, Click is omitted from the order of events leading to the DblClick event.

If the return value of *Cancel* is **True** when the user clicks twice, the control ignores the second click. This is useful if the second click reverses the effect of the first, such as double-clicking a toggle button. The *Cancel* argument allows your HTML Layout to ignore the second click, so clicking or double-clicking the button has the same effect.

Enter, Exit Events

See Also   Example   Applies To

Enter occurs before a control actually receives the focus from a control on the same HTML Layout. Exit occurs immediately before a control loses the focus to another control on the same HTML Layout.

Syntax

Private Sub *object*_**Enter( )**
Private Sub *object*_**Exit(** *Cancel* **As MSForms.ReturnBoolean)**

The **Enter** and **Exit** event syntaxes have these parts:

Part	Description
object	Required. A valid object name.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates the application should handle the event and the focus should remain on the current control.

Remarks

The Enter and Exit events are similar to the GotFocus and LostFocus events in VBScript. Unlike GotFocus and LostFocus, the Enter and Exit events don't occur when an HTML Layout receives or loses the focus.

For example, suppose you select the check box that initiates the Enter event. If you then select another control in the same HTML Layout, the Exit event will be initiated for the check box (because the focus is moving to a different object in the same HTML Layout) and the Enter event will occur for the second control on the HTML Layout.

Because the Enter event occurs before the focus moves to a particular control, you can use an Enter event procedure to display instructions. For example, you could use an event procedure to display a small HTML Layout or message box identifying the type of data the control typically contains.

Note   To prevent the control from losing focus, assign **True** to the *Cancel* argument of the Exit event.

MouseDown, MouseUp Events

See Also Example Applies To

Occur when the user clicks a mouse button. MouseDown occurs when the user presses the mouse button; MouseUp occurs when the user releases the mouse button.

Syntax

For TabStrip

Private Sub *object*_**MouseDown(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

Private Sub *object*_**MouseUp(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

For other controls

Private Sub *object*_**MouseDown(ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

Private Sub *object*_**MouseUp(ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

The **MouseDown** and **MouseUp** event syntaxes have these parts:

Part	Description
object	Required. A valid object.
index	Required. The index of the tab in a **TabStrip** with the specified event.
Button	Required. An integer value that identifies which mouse button caused the event.
Shift	Required. The state of SHIFT, CTRL, and ALT.
X, Y	Required. The horizontal or vertical position, in points, from the left or top edge of the HTML Layout.

Settings

The settings for *Button* are:

Constant	Value	Description
fmButtonLeft	1	The left button was pressed.
fmButtonRight	2	The right button was pressed.
fmButtonMiddle	4	The middle button was pressed.

The settings for *Shift* are:

Value	Description
1	SHIFT was pressed.
2	CTRL was pressed.
3	SHIFT and CTRL were pressed.
4	ALT was pressed.
5	ALT and SHIFT were pressed.
6	ALT and CTRL were pressed.
7	ALT, SHIFT, and CTRL were pressed.

You can identify individual keyboard modifiers by using the following constants:

Constant	Value	Description
fmShiftMask	1	Mask to detect SHIFT.
fmCtrlMask	2	Mask to detect CTRL.
fmAltMask	4	Mask to detect ALT.

Remarks

For a **TabStrip**, the index argument identifies the tab that the user clicked. An index of –1 indicates the user did not click a tab. For example, if there are no tabs in the upper-right corner of the control, clicking in the upper-right corner sets the index to –1.

For an HTML Layout, the user can generate MouseDown and MouseUp events by pressing and releasing a mouse button in a blank area, record selector, or scroll bar on the HTML Layout.

The sequence of mouse-related events is:

1. MouseDown
2. MouseUp
3. Click
4. DblClick
5. MouseUp

MouseDown or MouseUp event procedures specify actions that occur when a mouse button is pressed or released. MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

If a mouse button is pressed while the pointer is over an HTML Layout or control, that object will "capture" the mouse and receive all mouse events up to and including the last MouseUp event. This implies that the *X*, *Y* mouse-pointer coordinates returned by a mouse event may not always be within the boundaries of the object that receives them.

If mouse buttons are pressed in succession, the object that captures the mouse will receive all successive mouse events until all buttons are released.

Use the *Shift* argument to identify the state of SHIFT, CTRL, and ALT when the MouseDown or MouseUp event occurred. For example, if both CTRL and ALT are pressed, the value of *Shift* will be 6.

MouseEnter, MouseExit Events

See Also Example Applies To

MouseEnter occurs when the mouse pointer is moved over the control. MouseExit occurs when the mouse pointer is moved off of the control.

Syntax

Private Sub *object*_**MousEnter( )**
Private Sub *object*_**MouseExit(** *Cancel* **As Boolean)**

The **MouseEnter** and **MouseExit** event syntax has these parts:

Part	**Description**
object	Required. A valid object name.
Cancel	Required. Event status. **False** indicates that the control should handle the event (default). **True** indicates that the application should handle the event and the focus should remain at the current control.

Remarks

You can use the MouseEnter and MouseExit events to make something interesting happen, like playing sound files, when the mouse pointer hovers over an object.

MouseMove Event

See Also Example Applies To

Occurs when the user moves the mouse.

Syntax

For TabStrip

Private Sub *object*_**MouseMove(** *index* **As Long**, **ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

For other controls

Private Sub *object*_**MouseMove(ByVal** *Button* **As fmButton**, **ByVal** *Shift* **As fmShiftState**, **ByVal** *X* **As Single**, **ByVal** *Y* **As Single)**

The **MouseMove** event syntax has these parts:

Part	Description
object	Required. A valid object name.
index	Required. The index of the tab in a **TabStrip** associated with this event.
Button	Required. An integer value that identifies the state of the mouse buttons.
Shift	Required. Specifies the state of SHIFT, CTRL, and ALT.
X, Y	Required. The horizontal or vertical position, measured in points, from the left or top edge of the control.

Settings

The *index* argument specifies which tab was clicked over. A –1 designates that the user did not click any of the tabs.

The settings for *Button* are:

Value	Description
0	No button is pressed.
1	The left button is pressed.
2	The right button is pressed.
3	The right and left buttons are pressed.
4	The middle button is pressed.
5	The middle and left buttons are pressed.
6	The middle and right buttons are pressed.
7	All three buttons are pressed.

The settings for *Shift* are:

Value	Description
1	SHIFT was pressed.
2	CTRL was pressed.
3	SHIFT and CTRL were pressed.
4	ALT was pressed.
5	ALT and SHIFT were pressed.
6	ALT and CTRL were pressed.
7	ALT, SHIFT, and CTRL were pressed.

You can identify individual keyboard modifiers by using the following constants:

Constant	Value	Description
fmShiftMask	1	Mask to detect SHIFT.
fmCtrlMask	2	Mask to detect CTRL.
fmAltMask	4	Mask to detect ALT.

Remarks

The MouseMove event applies to HTML Layouts, controls on an HTML Layout, and labels.

MouseMove events are generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

Moving an HTML Layout can also generate a MouseMove event even if the mouse is stationary. MouseMove events are generated when the HTML Layout moves underneath the pointer. If an event procedure moves an HTML Layout in response to a MouseMove event, the event can continually generate (cascade) MouseMove events.

If two controls are very close together, and you move the mouse pointer quickly over the space between them, the MouseMove event might not occur for that space. In such cases, you might need to respond to the MouseMove event in both controls.

You can use the value returned in the *Button* argument to identify the state of the mouse buttons.

Use the *Shift* argument to identify the state of SHIFT, CTRL, and ALT when the MouseMove event occurred. For example, if both CTRL and ALT are pressed, the value of *Shift* will be 6.

Note You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

onLoad Event

See Also Example Applies To

OnLoad occurs when the entire contents of the HTML Layout (.ALX file) are created and before the Window onLoad event occurs.

Syntax

sub *object.***onLoad=***function-name*

Part	**Description**
object	Required. The filename of the HTML Layout object (.ALX file).
function-name	An object expression which evaluates to a scripting function.

Remarks

The order of onLoad events is the HTML Layout, then the Window onLoad event. The Window onLoad event only fires after the entire contents of the window have been rendered.

You can script the Window onLoad event from HTML but not from within the .ALX. Therefore, when editing an .ALX file, the Script Wizard does not display the Window Load/Unload events.

There is no HTML Layout onUnLoad event.

Item Method

See Also Example Applies To

Returns a member of a collection, either by position or by name.

Syntax

Set *Object* = *object*.**Item(** *collectionindex***)**

The **Item** method syntax has these parts:

Part	Description
object	Required. A valid object.
collectionindex	Required. A member's position, or index, within a collection.

Settings

The *collectionindex* can be either a string or an integer. If it is a string, it must be a valid member name. If it is an integer, the minimum value is 0 and the maximum value is one less than the number of items in the collection.

Remarks

If an invalid index or name is specified, an error occurs.

Move Method

See Also Example Applies To

Moves an HTML Layout or control, or moves all the controls in the **Controls** collection.

Syntax

object.**Move(** [*Left* [, *Top* [, *Width* [, *Height*]]]]**)**

The **Move** method syntax has these parts:

Part	Description
object	Required. A valid object name.
Left	Optional. Single-precision value, in points, indicating the horizontal coordinate for the left edge of the object.
Top	Optional. Single-precision value, in points, that specifies the vertical coordinate for the top edge of the object.
Width	Optional. Single-precision value, in points, indicating the width of the object.
Height	Optional. Single-precision value, in points, indicating the height of the object.

Settings

The maximum and minimum values for the *Left*, *Top*, *Width*, *Height*, *X*, and *Y* arguments vary from one application to another.

Remarks

For an HTML Layout or control, you can move a selection to a specific location relative to the edges of the HTML Layout that contains the selection.

You can use named arguments, or you can enter the arguments by position. If you use named arguments, you can list the arguments in any order. If not, you must enter the arguments in the order shown, using commas to indicate the relative position of arguments you do not specify. Any unspecified arguments remain unchanged.

ZOrder Method

See Also Example Applies To

Places the object at the front or back of the z-order.

Syntax

object.**ZOrder(** [*zPosition*]**)**

The **ZOrder** method syntax has these parts:

Part	Description
object	Required. A valid object.
zPosition	Optional. A control's position, front or back, in the container's z-order.

Settings

The settings for *zPosition* are:

Constant	Value	Description
fmTop	0	Places the control at the front of the z-order. The control appears on top of other controls (default).
fmBottom	1	Places the control at the back of the z-order. The control appears underneath other controls.

Remarks

The z-order determines how windows and controls are stacked when they are presented to the user. Items at the back of the z-order are overlaid by closer items; items at the front of the z-order appear to be on top of items at the back. When the *zPosition* argument is omitted, the object is brought to the front.

In design time, the **Bring to Front** or **Send to Back** commands set the z-order. **Bring to Front** is equivalent to using the **ZOrder** method and putting the object at the front of the z-order. **Send to Back** is equivalent to using **ZOrder** and putting the object at the back of the z-order.

This method does not affect the content or sequence of the controls in the **Controls** collection.

Note You can't **Undo** or **Redo** layering commands such as **Send to Back** or **Bring to Front**. For example, if you select an object and click **Move Backward** on the shortcut menu, you won't be able to Undo or redo that action.

AutoSize Property

See Also Example Applies To

Specifies whether an object automatically resizes to display its entire contents.

Syntax

object.**AutoSize** [= *Boolean*]

The **AutoSize** property syntax has these parts:

Part	Description
object	Required. A valid object.
Boolean	Optional. Specifies whether the control is resized.

Settings

The settings for *Boolean* are:

Value	Description
True	Automatically resizes the control to display its entire contents.
False	Keeps the size of the control constant. Contents are cropped when they exceed the area of the control (default).

Remarks

For controls with captions, the **AutoSize** property specifies whether the control automatically adjusts to display the entire caption.

For controls without captions, this property specifies whether the control automatically adjusts to display the information stored in the control. In a **ComboBox**, for example, setting **AutoSize** to **True** automatically sets the width of the display area to match the length of the current text.

For a single-line text box, setting **AutoSize** to **True** automatically sets the width of the display area to the length of the text in the text box.

For a multiline text box that contains no text, setting **AutoSize** to **True** automatically displays the text as a column. The width of the text column is set to accommodate the widest letter of that font size. The height of the text column is set to display the entire text of the **TextBox**.

For a multiline text box that contains text, setting **AutoSize** to **True** automatically enlarges the **TextBox** vertically to display the entire text. The width of the **TextBox** does not change.

Note If you manually change the size of a control while **AutoSize** is **True**, the manual change will override the size previously set by **AutoSize**.

BackColor Property

See Also Example Applies To

Specifies the background color of the object.

Syntax

object.**BackColor** [= *Long*]

The **BackColor** property syntax has these parts:

Part	Description
object	Required. A valid object.
Long	Optional. A value or constant that determines the background color of an object.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from 0 to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

Remarks

You can see the background color of an object only if the **BackStyle** property is set to **fmBackStyleOpaque**.

BackStyle Property

See Also Example Applies To

Returns or sets the background style for an object.

Syntax

object.**BackStyle** [= *fmBackStyle*]

The **BackStyle** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmBackStyle	Optional. Specifies the control background.

Settings

The settings for *fmBackStyle* are:

Constant	Value	Description
fmBackStyleTransparent	0	The background is transparent.
fmBackStyleOpaque	1	The background is opaque (default).

Remarks

The **BackStyle** property determines whether a control is transparent. If **BackStyle** is **fmBackStyleOpaque**, the control is not transparent and you cannot see anything behind the control on an HTML Layout. If **BackStyle** is **fmBackStyleTransparent**, you can see through the control and look at anything on the HTML Layout located behind the control.

Note The **BackStyle** property does not affect the transparency of bitmaps. You must use a picture editor to make a bitmap transparent. Not all controls support transparent bitmaps.

Bold, Italic, Size, StrikeThrough, Underline, Weight Properties

See Also Example Applies To

Specifies the visual attributes of text on a displayed or printed HTML Layout.

Syntax

object.**Bold** [= *Boolean*]
object.**Italic** [= *Boolean*]
object.**Size** [= *Currency*]
object.**StrikeThrough** [= *Boolean*]
object.**Underline** [= *Boolean*]
object.**Weight** [= *Integer*]

The **Bold**, **Italic**, **Size**, **StrikeThrough**, **Underline**, and **Weight** property syntaxes have these parts:

Part	Description
object	Required. A valid object name.
Boolean	Optional. Specifies the font style.
Currency	Optional. A number indicating the font size.
Integer	Optional. Specifies the font style.

The settings for *Boolean* are:

Value	Description
True	The text has the specified attribute (that is bold, italic, size, strikethrough or underline marks, or weight).
False	The text does not have the specified attribute (default).

The **Weight** property accepts values from 0 to 1000. A value of zero allows the system to pick the most appropriate weight. A value from 1 to 1000 indicates a specific weight, where 1 represents the lightest type and 1000 represents the darkest type.

Remarks

These properties define the visual characteristics of text. The **Bold** property determines whether text is normal or bold. The **Italic** property determines whether text is normal or italic. The **Size** property determines the height, in points, of displayed text. The **Underline** property determines whether text is underlined. The **StrikeThrough** property determines whether the text appears with strikethrough marks. The **Weight** property determines the darkness of the type.

There may be a difference between how a font appears on screen and how it looks printed, depending on your computer and printer. If you select a font that your system can't display with the specified attribute or that isn't installed, Windows substitutes a similar font. The substitute font will be as similar as possible to the font originally requested.

Changing the value of **Bold** also changes the value of **Weight**. Setting **Bold** to **True** sets **Weight** to 700; setting **Bold** to **False** sets **Weight** to 400. Conversely, setting **Weight** to anything over 550 sets **Bold** to **True**; setting **Weight** to 550 or less sets **Bold** to **False**.

The default point size is determined by the operating system.

BorderColor Property

See Also Example Applies To

Specifies the color of a control's border.

Syntax

object.**BorderColor** [= *Long*]

The **BorderColor** property syntax has these parts:

Part	Description
object	Required. A valid object.
Long	Optional. A value or constant that determines the border color of a control.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from 0 to 255. For example, you can specify teal blue as the integer value 4966415 or as RGB color component values 15, 200, 75.

Remarks

To use the **BorderColor** property, the **BorderStyle** property must be set to a value other than **fmBorderStyleNone**.

BorderStyle uses **BorderColor** to define the border colors. The **SpecialEffect** property uses system colors exclusively to define its border colors. For Windows operating systems, system color settings are part of the **Control Panel** and are found in the **Appearance** tab of the **Display** folder. In Windows NT 3.51, system color settings are stored in the **Color** folder of the **Control Panel**.

BorderStyle Property

See Also Example Applies To

Specifies the type of border used by a control.

Syntax

object.**BorderStyle** [= *fmBorderStyle*]

The **BorderStyle** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmBorderStyle	Optional. Specifies the border style.

Settings

The settings for *fmBorderStyle* are:

Constant	Value	Description
fmBorderStyleNone	0	The control has no visible border line.
fmBorderStyleSingle	1	The control has a single-line border (default).

The default value for a **ComboBox**, **Label**, **ListBox** or **TextBox** is 0 (*None*). The default value for an **Image** is 1 (*Single*).

Remarks

You can use either **BorderStyle** or **SpecialEffect** to specify the border for a control, but not both. If you specify a value other than zero for one of these properties, the system sets the value of the other property to zero. For example, if you set **BorderStyle** to **fmBorderStyleSingle**, the system sets **SpecialEffect** to zero (*Flat*). If you specify a value other than zero for **SpecialEffect**, the system sets **BorderStyle** to zero.

BorderStyle uses **BorderColor** to define the colors of its borders.

CodeBase Property

See Also Example Applies To

Specifies the URL of a control's COM object.

Remarks

The **CodeBase** property makes it possible to automatically download ActiveX controls from a server to a user's machine.

The CodeBase property supports the following file types:

File type	Description
PE (portable executable)	The PE (for example, .ocx, .dll, .exe) is downloaded, installed, and registered automatically if the control is not already registered on the user's computer. This is the simplest way to package a single-file ActiveX control, but it does not use file compression and isn't platform independent except with HTTP.
cab (cabinet)	The .cab file contains one or more files, all of which are downloaded together in a single compressed cabinet file. One file in the cabinet is an .inf file providing further installation information. The .inf file may refer to files in the .cab as well as to files at other URLs.
inf (installation information)	The stand-alone .inf file specifies various files that need to be downloaded and set up for an .ocx to run. The syntax of the .inf file supports URLs pointing to files to download as well as platform independence (by enumerating files for various platforms). This mechanism provides platform independence for non-HTTP servers.

For specifics about creating PE, .cab, and .inf files and for the latest information about Internet Component Download, go to

http://www.microsoft.com/intdev/signcode/codedwld.htm

on the Internet.

Note The **CodeBase** property can be set only at design time. It can't be set at run time.

DrawBuffer Property

See Also   Example   Applies To

Specifies the suggested number of pixels set aside for off-screen memory in rendering an HTML Layout.

Syntax

object.**DrawBuffer** [= *value*]

Part	**Description**
object	Required. A valid object name.
value	An integer between 16,000 to 1,048,576 equal to the maximum number of pixels the object will render off-screen. The default value is 32,000, which covers, for example, an area of 80x400 pixels.

Remarks

The **DrawBuffer** property specifies the maximum number of pixels that can be drawn at one time as the display repaints. The actual memory used by the HTML Layout depends on the screen resolution of the display. If you set a large value for **DrawBuffer**, performance will be slower. A large buffer helps when several large images overlap.

The **DrawBuffer** property cannot be set from the Properties window in your .alx file. You can set **DrawBuffer** in Script Wizard by selecting the HTML Layout **onLoad** event.

Enabled Property

See Also Example Applies To

Specifies whether a control can receive the focus and respond to user-generated events.

Syntax

object.**Enabled** [= *Boolean*]

The **Enabled** property syntax has these parts:

Part	Description
object	Required. A valid object.
Boolean	Optional. Specifies whether the object can respond to user-generated events.

Settings

The settings for *Boolean* are:

Value	Description
True	The control can receive the focus and respond to user-generated events, and is accessible through code (default).
False	The user cannot interact with the control by using the mouse, keystrokes, accelerators, or hot keys. The control is generally still accessible through code.

Remarks

Use the **Enabled** property to enable and disable controls. A disabled control appears dimmed, while an enabled control does not. Also, if a control displays a bitmap, the bitmap is dimmed whenever the control is dimmed. If **Enabled** is **False** for an **Image**, the control will not initiate events but it will also not appear dimmed.

The **Enabled** and **Locked** properties work together to achieve the following effects:

- If **Enabled** and **Locked** are both **True**, the control can receive focus and it will appear normally (not dimmed) in the HTML Layout. The user can copy, but not edit, data in the control.
- If **Enabled** is **True** and **Locked** is **False**, the control can receive focus and it will appear normally in the HTML Layout. The user can copy and edit data in the control.
- If **Enabled** is **False** and **Locked** is **True**, the control cannot receive focus and it will appear dimmed in the HTML Layout. The user can neither copy nor edit data in the control.
- If **Enabled** and **Locked** are both **False**, the control cannot receive focus and it will appear dimmed in the HTML Layout. The user can neither copy nor edit data in the control.

You can combine the settings of the **Enabled** and the **TabStop** properties to prevent the user from selecting a command button with TAB, while still allowing the user to click the button. Setting **TabStop** to **False** means the command button will not appear in the tab order. However, if **Enabled** is **True**, then the user can still click the command button, as long as **TakeFocusOnClick** is set to **True**.

When the user tabs into an enabled **TabStrip**, the first page or tab in the control receives the focus. If the first page or tab of a **TabStrip** is disabled, the first enabled page or tab of that control will receive the focus. If all pages or tabs of a or **TabStrip** are disabled, the control will be disabled and will not be able to receive the focus.

ForeColor Property

See Also Example Applies To

Specifies the foreground color of an object.

Syntax

object.**ForeColor** [= *Long*]

The **ForeColor** property syntax has these parts:

Part	Description
object	Required. A valid object.
Long	Optional. A value or constant that determines the foreground color of an object.

Settings

You can use any integer that represents a valid color. You can also specify a color by using the RGB function with red, green, and blue color components. The value of each color component is an integer that ranges from 0 to 255. For example, you can specify teal blue as the integer value 4966415 or as red, green, and blue color components 15, 200, 75.

Remarks

Use the **ForeColor** property for controls on HTML Layouts to make them easy to read or to convey a special meaning. For example, if a text box reports the number of units in stock, you can change the color of the text when the value falls below the reorder level.

For a **ScrollBar** or **SpinButton**, the **ForeColor** property sets the color of the arrows. For a **Font** object, the **ForeColor** property determines the color of the text.

Height, Width Properties

See Also Example Applies To

The height or width, in points, of an object.

Syntax

object.**Height** [= *Single*]
object.**Width** [= *Single*]

The **Height** and **Width** property syntaxes have these parts:

Part	Description
object	Required. A valid object.
Single	Optional. A numeric expression specifying the dimensions of an object.

Remarks

The **Height** and **Width** properties are automatically updated when you move or size a control. If you change the size of a control, the **Height** or **Width** property will store the new height or width. If you specify a setting for the **Left** or **Top** property that is less than zero, that value will be used to calculate the height or width of the control, but a portion of the control will not be visible on the HTML Layout.

If you move a control from one part of an HTML Layout to another, the setting of **Height** or **Width** will change only if you size the control as you move it. The settings of the control's **Left** and **Top** properties will change to reflect the control's new position relative to the edges of the HTML Layout that contains it.

The value assigned to **Height** or **Width** must be greater than or equal to zero. For most systems, the recommended range of values is from 0 to +32,767. Higher values may also work depending on your system configuration.

ID Property

See Also Example Applies To

Specifies the name of a control or an object, or the name of a font to associate with a **Font** object.

Syntax

For Font
  Font.**ID** [= *String*]
For all other controls and objects
  object.**ID** [= *String*]

The **ID** property syntax has these parts:

Part	Description
object	Required. A valid object.
String	Optional. The name you want to assign to the font or control.

Settings

Guidelines for assigning a string to the **ID** property, such as the maximum length of the name, vary from one application to another.

Remarks

For objects, the default value of **ID** consists of the object's class name followed by an integer. For example, the default name for the first **TextBox** you place on an HTML Layout is TextBox1. The default name for the second **TextBox** is TextBox2.

You can set the **ID** property for a control from the control's Properties window or, for controls added at run time, by using program statements. If you add a control at design time, you cannot modify its **ID** property at run time.

Each control added to an HTML Layout at design time must have a unique name.

For **Font** objects, the **ID** property identifies a particular typeface to use in the text portion of a control, object, or HTML Layout. The font's appearance on screen and in print may differ, depending on your computer and printer. If you select a font that your system can't display or that isn't installed, Windows will substitute a similar font.

Left, Top Properties

See Also Example Applies To

The distance between a control and the left or top edge of the HTML Layout that contains it.

Syntax

object.**Left** [= *Single*]
object.**Top** [= *Single*]

The **Left** and **Top** property syntaxes have these parts:

Part	Description
object	Required. A valid object.
Single	Optional. A numeric expression specifying the coordinates of an object.

Settings

Setting the **Left** or **Top** property to zero places the control's edge at the left or top edge of its container.

Remarks

For most systems, the recommended range of values for **Left** and **Top** is from -32,767 to +32,767. Other values may also work depending on your system configuration. For a **ComboBox**, values of **Left** and **Top** apply to the text portion of the control, not to the list portion. When you move or size a control, its new **Left** setting is automatically entered in the Properties window. When you print an HTML Layout, the control's horizontal or vertical location is determined by its **Left** or **Top** setting.

MouseIcon Property

See Also Example Applies To

Assigns a custom icon to an object.

Syntax

object.**MouseIcon** = **LoadPicture(** *pathname* **)**

The **MouseIcon** property syntax has these parts:

Part	Description
object	Required. A valid object.
pathname	Required. A string expression specifying the path and filename of the file containing the custom icon.

Remarks

The **MouseIcon** property is valid when the **MousePointer** property is set to 99. The mouse icon of an object is the image that appears when the user moves the mouse across that object.

To assign an image for the mouse pointer, you can either assign a picture to the **MouseIcon** property or load a picture from a file using the **LoadPicture** function.

MousePointer Property

See Also Example Applies To

Specifies the type of pointer displayed when the user positions the mouse over a particular object.

Syntax

object.**MousePointer** [= *fmMousePointer*]

The **MousePointer** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmMousePointer	Optional. The shape you want for the mouse pointer.

Settings

The settings for *fmMousePointer* are:

Constant	Value	Description
fmMousePointerDefault	0	Standard pointer. The image is determined by the object (default).
fmMousePointerArrow	1	Arrow.
fmMousePointerCross	2	Cross-hair pointer.
fmMousePointerIBeam	3	I-beam.
fmMousePointerSizeNESW	6	Double arrow pointing northeast and southwest.
fmMousePointerSizeNS	7	Double arrow pointing north and south.
fmMousePointerSizeNWSE	8	Double arrow pointing northwest and southeast.
fmMousePointerSizeWE	9	Double arrow pointing west and east.
fmMousePointerUpArrow	10	Up arrow.
fmMousePointerHourglass	11	Hourglass.
fmMousePointerNoDrop	12	"Not" symbol (circle with a diagonal line) on top of the object being dragged. Indicates an invalid drop target.
fmMousePointerAppStarting	13	Arrow with an hourglass.
fmMousePointerHelp	14	Arrow with a question mark.
fmMousePointerSizeAll	15	Size all cursor (arrows pointing north, south, east, and west).
fmMousePointerCustom	99	Uses the icon specified by the **MouseIcon** property.

Remarks

Use the **MousePointer** property when you want to indicate changes in functionality as the mouse pointer passes over controls on an HTML Layout. For example, the hourglass setting (11) is useful to indicate that the user must wait for a process or operation to finish.

Some icons vary depending on system settings, such as the icons associated with desktop themes.

PictureAlignment Property

See Also Example Applies To

Specifies the location of a background picture.

Syntax

object.**PictureAlignment** [= *fmPictureAlignment*]

The **PictureAlignment** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmPictureAlignment	Optional. The position where the picture aligns with the control.

Settings

The settings for *fmPictureAlignment* are:

Constant	Value	Description
fmPictureAlignmentTopLeft	0	The top-left corner.
fmPictureAlignmentTopRight	1	The top-right corner.
fmPictureAlignmentCenter	2	The center.
fmPictureAlignmentBottomLeft	3	The bottom-left corner.
fmPictureAlignmentBottomRight	4	The bottom-right corner.

Remarks

The **PictureAlignment** property identifies which corner of the picture is the same as the corresponding corner of the control or container where the picture is used.

For example, setting **PictureAlignment** to **fmPictureAlignmentTopLeft** means that the top-left corner of the picture coincides with the top-left corner of the control or container. Setting **PictureAlignment** to **fmPictureAlignmentCenter** positions the picture in the middle, relative to the height as well as the width of the control or container.

If you tile an image on a control or container, the setting of **PIctureAlignment** will affect the tiling pattern. For example, if **PictureAlignment** is set to **fmPictureAlignmentUpperLeft**, the first copy of the image will be placed in the upper-left corner of the control or container and additional copies will be tiled from left to right across each row. If **PictureAlignment** is **fmPictureAlignmentCenter**, the first copy of the image will be placed at the center of the control or container, additional copies will be placed to the left and right to complete the row, and additional rows will be added to fill the control or container.

Note Setting the **PictureSizeMode** property to **fmSizeModeStretch** overrides **PictureAlignment**. When **PictureSizeMode** is set to **fmSizeModeStretch**, the picture fills the entire control or container.

PicturePath Property

See Also Example Applies To

Specifies the URL of the picture to display on **Image** control.

Syntax

object.**PicturePath** = *URL*

The **PicturePath** property syntax has these parts:

Part	Description
object	Required. A valid object.
URL	Required. The URL of a picture file.

Remarks

PicturePath requires a complete URL. It does not support a UNC path.

PictureSizeMode Property

See Also Example Applies To

Specifies how to display the background picture on a control, HTML Layout, or HTML page.

Syntax

object.**PictureSizeMode** [= *fmPictureSizeMode*]

The **PictureSizeMode** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmPictureSizeMode	Optional. The action to take if the picture and the HTML Layout or HTML page that contains it are not the same size.

Settings

The settings for *fmPictureSizeMode* are:

Constant	Value	Description
fmPictureSizeModeClip	0	Crops any part of the picture that is larger than the HTML Layout or HTML page (default).
fmPictureSizeModeStretch	1	Stretches the picture to fill the HTML Layout or HTML page. This setting distorts the picture in either the horizontal or vertical direction.
fmPictureSizeModeZoom	3	Enlarges the picture, but does not distort the picture in either the horizontal or vertical direction.

Remarks

The **fmPictureSizeModeClip** setting indicates you want to show the picture in its original size and scale. If the HTML Layout or HTML page is smaller than the picture, this setting will show only the part of the picture that fits within the HTML Layout or HTML page.

The **fmPictureSizeModeStretch** and **fmPictureSizeModeZoom** settings both enlarge the image, but **fmPictureSizeModeStretch** causes distortion. The **fmPictureSizeModeStretch** setting enlarges the image horizontally and vertically until the image reaches the corresponding edges of the container or control. The **fmPictureSizeModeZoom** setting enlarges the image until it reaches either the horizontal or vertical edges of the container or control. If the image reaches the horizontal edges first, any remaining distance to the vertical edges will remain blank. If it reaches the vertical edges first, any remaining distance to the horizontal edges will remain blank.

PictureTiling Property

See Also Example Applies To

Lets you tile a picture in an image control.

Syntax

object.**PictureTiling** [= *Boolean*]

The **PictureTiling** property syntax has these parts:

Part	Description
object	Required. A valid object.
Boolean	Optional. Specifies whether a picture is repeated across a background.

Settings

The settings for *Boolean* are:

Value	Description
True	The picture is tiled across the background.
False	The picture is not tiled across the background (default).

Remarks

You can tile an image on an HTML Layout by drawing the **Image** the same size as the HTML Layout.

The tiling pattern depends on the current setting of the **PictureAlignment** and **PictureSizeMode** properties. For example, if **PictureAlignment** is set to **fmPictureAlignmentTopLeft**, the tiling pattern will start at the upper-left, repeating the picture across and down the height of the **Image**. If **PictureSizeMode** is set to **fmPictureSizeModeClip**, the tiling pattern will crop the last tile if it doesn't completely fit within the **Image**.

SpecialEffect Property

See Also Example Applies To

Specifies the visual appearance of an object.

Syntax

For CheckBox, OptionButton, ToggleButton
object.**SpecialEffect** [= *fmButtonEffect*]
For other controls
object.**SpecialEffect** [= *fmSpecialEffect*]

The **SpecialEffect** property syntax has these parts:

Part	Description
object	Required. A valid object.
fmButtonEffect	Optional. The desired visual appearance for a **CheckBox**, **OptionButton**, or **ToggleButton**.
fmSpecialEffect	Optional. The desired visual appearance of an object other than a **CheckBox**, **OptionButton**, or **ToggleButton**.

Settings

The settings for *fmSpecialEffect* are:

Constant	Value	Description
fmSpecialEffectFlat	0	Object appears flat, distinguished from the surrounding form by a border, a change of color, or both. Default for **Image** and **Label**, valid for all controls.
fmSpecialEffectRaised	1	Object has a highlight on the top and left and a shadow on the bottom and right. Not valid for check boxes or option buttons.
fmSpecialEffectSunken	2	Object has a shadow on the top and left and a highlight on the bottom and right. The control and its border appear to be carved into the form that contains them. Default for **CheckBox** and **OptionButton**, valid for all controls (default).
fmSpecialEffectEtched	3	Border appears to be carved around the edge of the control. Not valid for check boxes or option buttons.
fmSpecialEffectBump	6	Object has a ridge on the bottom and right and appears flat on the top and left. Not valid for check boxes or option buttons.

For a **Frame**, the default value is *Sunken*.

Note that only *Flat* and *Sunken* (0 and 2) are acceptable values for **CheckBox**, **OptionButton**, and **ToggleButton**. All values listed are acceptable for other controls.

Remarks

You can use either the **SpecialEffect** or the **BorderStyle** property to specify the edging for a control, but not both. If you specify a value other than zero for one of these properties, the system sets the value of the other property to zero. For example, if you set **SpecialEffect** to **fmSpecialEffectRaised**, the system sets **BorderStyle** to zero (**fmBorderStyleNone**).

For a **Frame**, **BorderStyle** is ignored if **SpecialEffect** is **fmSpecialEffectFlat**.

SpecialEffect uses the system colors to define its borders.

Note Although the **SpecialEffect** property exists on the **ToggleButton**, the property is disabled. You cannot set or return a value for this property on the **ToggleButton**.

TabIndex Property

See Also   Example   Applies To

Specifies the position of a single object in the HTML Layout's tab order.

Syntax

object.**TabIndex** [= *Integer*]

The **TabIndex** property syntax has these parts:

Part	Description
object	Required. A valid object.
Integer	Optional. An integer from 0 to one less than the number of controls on the HTML Layout that have a **TabIndex** property. Assigning a **TabIndex** value of less than 0 generates an error. If you assign a **TabIndex** value greater than the largest index value, the system resets the value to the maximum allowable value.

Remarks

The index value of the first object in the tab order is zero.

TabStop Property

See Also Example Applies To

Indicates whether an object can receive focus when the user tabs to it.

Syntax

object.**TabStop** [= *Boolean*]

The **TabStop** property syntax has these parts:

Part	Description
object	Required. A valid object.
Boolean	Optional. Specifies whether the object is a tab stop.

Settings

The settings for *Boolean* are:

Value	Description
True	Designates the object as a tab stop (default).
False	Bypasses the object when the user is tabbing, although the object still holds its place in the actual tab order, as determined by the **TabIndex** property.

Remarks

The **TabStop** property can be set only at design time.

Visible Property

See Also Example Applies To

Specifies whether a control is visible or hidden.

Syntax

object.**Visible** [= *Boolean*]

The **Visible** property syntax has these parts:

Part	Description
object	Required. A valid object.
Boolean	Optional. Whether the object is visible or hidden.

Settings

The settings for *Boolean* are:

Value	Description
True	Object is visible (default).
False	Object is hidden.

Remarks

To hide an object at startup, set the **Visible** property to **False** at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

All controls are visible at design time.

Adding Links via Colored Labels Example

This example uses **Labels** to provide links to other sites rather than underlined hypertext links. This text describes the process for adding one **Label**. This example could be extended to use **Image** and **Hot Spot** controls.

The following control and corresponding property values should be set:

- Add **Label1**
  - Select a background color and foreground color. Set Caption (for example "Microsoft").

For the **Label1** MouseDown event, add the following code:

```
Window.location.href = "http://www.microsoft.com"
```

Mouse Down Event

AutoSize Property Example

The following example demonstrates the effects of the **AutoSize** property with a single-line **TextBox** and a multiline **TextBox**. The user can enter text into either **TextBox** and turn **AutoSize** on or off independently of the contents of the **TextBox**. This code sample also uses the **Text** property.

To use this example, copy this sample code to the Declarations portion of a form. Make sure that the form contains:

- Two **TextBox** controls named TextBox1 and TextBox2.
- A **ToggleButton** named ToggleButton1.

```
Private Sub UserForm_Initialize()
    TextBox1.Text = "Single-line TextBox. Type your text here."

    TextBox2.MultiLine = True
    TextBox2.Text = "Multi-line TextBox. Type your text here. Use
CTRL+ENTER to start a new line."

    ToggleButton1.Value = True
    ToggleButton1.Caption = "AutoSize On"
    TextBox1.AutoSize = True
    TextBox2.AutoSize = True
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "AutoSize On"
        TextBox1.AutoSize = True
        TextBox2.AutoSize = True
    Else
        ToggleButton1.Caption = "AutoSize Off"
        TextBox1.AutoSize = False
        TextBox2.AutoSize = False
    End If
End Sub
```

Border, Color Enhancements Example

The following example demonstrates the **BorderStyle** and **SpecialEffect** properties, showing each border available through these properties. The example also demonstrates how to control color settings by using the **BackColor**, **BackStyle**, **BorderColor**, and **ForeColor** properties.

To use this example, copy this sample code to the Declarations portion of an HTML Layout. Make sure that the HTML Layout contains:

- Six **TextBox** controls named TextBox1 through TextBox6.
- Two **ToggleButton** controls named ToggleButton1 and ToggleButton2.

```
Private Sub UserLayout_Initialize()
' Initialize each TextBox with a border style or special effect,
' and foreground and background colors

' TextBox1 initially uses a borderstyle
TextBox1.Text = "BorderStyle-Single"
TextBox1.BorderStyle = fmBorderStyleSingle
TextBox1.BorderColor = RGB(255, 128, 128)   'Color - Salmon
TextBox1.ForeColor = RGB(255, 255, 0)       'Color - Yellow
TextBox1.BackColor = RGB(0, 128, 64)        'Color - Green #2

' TextBoxes 2 through 6 initially use special effects
TextBox2.Text = "Flat"
TextBox2.SpecialEffect = fmSpecialEffectFlat
TextBox2.ForeColor = RGB(64, 0, 0)          'Color - Brown
TextBox2.BackColor = RGB(0, 0, 255)         'Color - Blue

' Ensure the background style for TextBox2 is initially opaque.
TextBox2.BackStyle = fmBackStyleOpaque

TextBox3.Text = "Etched"
TextBox3.SpecialEffect = fmSpecialEffectEtched
TextBox3.ForeColor = RGB(128, 0, 255)       'Color - Purple
TextBox3.BackColor = RGB(0, 255, 255)       'Color - Cyan

'Define BorderColor for later use (when borderstyle=fmBorderStyleSingle)
TextBox3.BorderColor = RGB(0, 0, 0)         'Color - Black

TextBox4.Text = "Bump"
TextBox4.SpecialEffect = fmSpecialEffectBump
TextBox4.ForeColor = RGB(255, 0, 255)       'Color - Magenta
TextBox4.BackColor = RGB(0, 0, 100)         'Color - Navy blue

TextBox5.Text = "Raised"
TextBox5.SpecialEffect = fmSpecialEffectRaised
TextBox5.ForeColor = RGB(255, 0, 0)         'Color - Red
TextBox5.BackColor = RGB(128, 128, 128)     'Color - Gray

TextBox6.Text = "Sunken"
TextBox6.SpecialEffect = fmSpecialEffectSunken
TextBox6.ForeColor = RGB(0, 64, 0)          'Color - Olive
TextBox6.BackColor = RGB(0, 255, 0)         'Color - Green #1

ToggleButton1.Caption = "Swap styles"
ToggleButton2.Caption = "Transparent/Opaque background"
```

```
End Sub

Private Sub ToggleButton1_Click()

‘Swap borders between TextBox1 and TextBox3
If ToggleButton1.Value = True Then
   'Change TextBox1 from BorderStyle to Etched
   TextBox1.Text = "Etched"
   TextBox1.SpecialEffect = fmSpecialEffectEtched

   'Change TextBox3 from Etched to BorderStyle
   TextBox3.Text = "BorderStyle-Single"
   TextBox3.BorderStyle = fmBorderStyleSingle
Else
   'Change TextBox1 back to BorderStyle
   TextBox1.Text = "BorderStyle-Single"
   TextBox1.BorderStyle = fmBorderStyleSingle

   'Change TextBox3 back to Etched
   TextBox3.Text = "Etched"
   TextBox3.SpecialEffect = fmSpecialEffectEtched
End If
End Sub

Private Sub ToggleButton2_Click()

‘Set background to Opaque or Transparent
If ToggleButton2.Value = True Then
   'Change TextBox2 to a transparent background
   TextBox2.BackStyle = fmBackStyleTransparent
Else
   'Change TextBox2 back to opaque background
   TextBox2.BackStyle = fmBackStyleOpaque
End If

End Sub
```

Date and Time Example

This example uses a **CommandButton** Click event to update the **Caption** of a **Label** by choosing the value of two **CheckBoxes**.

The following controls and corresponding property values should be set:

- Add **CheckBox1**
  - Set **Caption** = "Show Date"
  - Set **Value** = 0
- Add **CheckBox2**
  - Set **Caption** = "Show Time"
  - Set **Value** = 0
- Add **Label1**
  - Set **Caption** = "Date and time displayed here"
  - Set **TextAlign** = Center
  - Set **BorderStyle** = Single
- Add **CommandButton1**
  - Set **Caption** = "Display"

For the **CommandButton1** Click event, add the following code:

```
Dim result
If CheckBox1.Value = True Then
    If CheckBox2.Value = True Then
        result = Date() & Chr(32) & Time()
    Else
        result = Date()
    End If
Else
    If CheckBox2.Value = True Then
        result = Time()
    Else
        result = "Date and time displayed here"
    End If
End If

Label1.Caption = result
```

Form Information Example

This example uses **OptionButtons** to change the background color. of an HTML Layout and a **CommandButton** to display the width and height of the HTML Layout in a dialog box.

The following controls and corresponding property values should be set:

- Add **OptionButton1**
  - Set **Caption** = "Red"
  - Set **Value** = 0
- Add **OptionButton2**
  - Set **Caption** = "Green"
  - Set **Value** = 0
- Add **OptionButton3**
  - Set **Caption** = "Blue"
  - Set **Value** = 0
- Add **CommandButton1**
  - Set **Caption** = "Size"

For the following events, add the corresponding code:

- For the **OptionButton1** Click event:
  ```
  Form.BackColor = RGB(255,0,0)
  ```
- For the **OptionButton2** Click event:
  ```
  Form.BackColor = RGB(0,255,0)
  ```
- For the **Optionbutton3** Click event:
  ```
  Form.BackColor = RGB(0,0,255)
  ```
- For the **CommandButton1** Click event:
  ```
  MsgBox("HTML Layout width = " & Form.Width & chr(13) & chr(10) & _
  "HTML Layoutheight = " & Form.Height)
  ```

Hello World Example

This example uses a **CommandButton** Click event to display the message "Hello World" in a dialog box.

The following control and corresponding property value should be set:

- Add **CommandButton1**
  - Set **Caption** = "Push"

For the **CommandButton1** Click event, add the following code:

```
MsgBox("Hello, World!")
```

Hide/Show Controls Example

This example demonstrates a method of hiding and showing **CommandButtons** on an HTML Layout.

The following controls and corresponding property values should be set:

- Add **CommandButton1**
  - Set Caption = "Show the other button"
- Add **CommandButton2**
  - Set Caption = "Bring back the first button"

For the following events, add the corresponding code:

- For the **CommandButton1** Click event:
  ```
  CommandButton2.Visible = True
  CommandButton1.Visible = False
  ```
- For the **CommandButton2** Click event:
  ```
  CommandButton1.Visible = True
  CommandButton2.Visible = False
  ```

Mouse Tracking Example

This example uses the MouseOver event for tracking mouse movement and updates a **Label** based on the mouse position.

The following controls and corresponding property values should be set:

- Add **Label1**
  - Set **Caption** = "Number One"
  - Set **BorderStyle** = "Single"
- Add **Label2**
  - Set **Caption** = "Number Two"
  - Set **BorderStyle** = "Single"
- Add **CommandButton1**
  - Set **Caption** = "Button 1"
- Add **Label3**
  - Set **Caption** = ""
  - Set **ID** = "lblDisplay"
  - Set **BorderStyle** = "Single"
  - Set **TextAlign** = "Center"

For the following events, add the corresponding code:

- For **Label1** MouseDown event:
  ```
  lblDisplay.Caption = "Mouse down number one"
  ```
- For **Label1** MouseMove event:
  ```
  lblDisplay.Caption = "Mouse moving over number one"
  ```
- For **Label2** MouseDown event:
  ```
  lblDisplay.Caption = "Mouse down number two"
  ```
- For **Label2** MouseMove event:
  ```
  lblDisplay.Caption = "Mouse moving over number two"
  ```
- For lblDisplay_MouseMove event:
  ```
  lblDisplay.Caption = "Mouse moving over display label"
  ```
- For **CommandButton1** MouseMove event:
  ```
  lblDisplay.Caption = "Mouse moving over command button"
  ```

Resizing an Image Example

This example demonstrates dynamically resizing an image.

The following controls and corresponding property values should be set:

- Add **Image1**
  - Assign the **PicturePath** property to some file. For example: `"file://c:\windows\test.bmp"`.
  - Set PictureSizeMode = "Stretch"
  - Size the HTML Layout to approximately twice as high and twice as wide as the image.
- Add **CommandButton1**
  - Set Caption = "Small"
- Add **CommandButton2**
  - Set Caption = "Medium"
- Add **CommandButton3**
  - Set Caption = "Large"

For the following events, add the corresponding code:

- For **CommandButton1** Click event:

```
Image1.Width = form.Width / 4
Image1.Height = form.Height / 4
Image1.Left = (form.Width/2) - (Image1.Width/2)
Image1.Top = (form.Height/2) - (Image1.Height/2)
```

- For **CommandButton2** Click event:

```
Image1.Width = form.Width / 2
Image1.Height = form.Height / 2
Image1.Left = (form.Width/2) - (Image1.Width/2)
Image1.Top = (form.Height/2) - (Image1.Height/2)
```

- For **CommandButton3** Click event:

```
Image1.Width = form.Width
Image1.Height = form.Height
Image1.Left = 0
Image1.Top = 0
```

Text Selection Properties Example

The following example tracks the selection-related properties (**SelLength**, **SelStart**, and **SelText**) that change as the user moves the insertion point and extends the selection using the keyboard. This example also uses the **Enabled** and **EnterFieldBehavior** properties.

To use this example, copy this sample code to the Declarations portion of an HTML Layout. Make sure that the HTML Layout contains:

1 One large **TextBox** named TextBox1.

2 Three **Label** objects named Label1, Label2, and Label3 in a column under TextBox1.

3 Three **TextBox** controls named TextBox2, TextBox3, and TextBox4 in a column to the right of Label1, Label2, and Label3.

4 A **CommandButton** named CommandButton1.

```
Private Sub ShowSelText()
   TextBox2.Text = TextBox1.SelStart
   TextBox3.Text = TextBox1.SelLength
   TextBox4.Text = TextBox1.SelText
End Sub


Private Sub TextBox1_Enter()
   ShowSelText
End Sub


Private Sub TextBox1_KeyDown(KeyCode As Integer, ByVal Shift As Integer)
   ShowSelText
End Sub


Private Sub TextBox1_KeyUp(KeyCode As Integer, ByVal Shift As Integer)
   ShowSelText
End Sub


Private Sub UserLayout_Initialize()
   TextBox1.MultiLine = True
   TextBox1.EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection

   TextBox1.Text = "SelText indicates the starting point of selected text,
or the insertion point if no text is selected." _
      & Chr$(10) & Chr$(13) & "The SelStart property is always valid, even
when the control does not have focus. Setting SelStart to a value less than
zero creates an error. " _
      & Chr$(10) & Chr$(13) & "Changing the value of SelStart cancels any
existing selection in the control, places an insertion point in the text,
and sets the SelLength property to zero."

   Label1.Caption = "Selection Start"
   Label1.AutoSize = True

   Label2.Caption = "Selection Length"
   Label2.AutoSize = True

   Label3.Caption = "Selected Text"
   Label3.AutoSize = True
   TextBox2.Enabled = False
   TextBox3.Enabled = False
   TextBox4.Enabled = False

   TextBox1.SetFocus
   TextBox1.CurLine = 0
   TextBox1.CurX = 317               'Current position settings are
himetric units.

   ShowSelText

   TextBox4.AutoSize = True
   TextBox4.MultiLine = True
   TextBox4.WordWrap = False
End Sub
```

TextBox Control Example

The following example tracks the **CurLine**, **CurTargetX**, and **CurX** property settings in a multiline **TextBox**. These settings change as the user moves the insertion point and extends the selection using the keyboard. This code sample also uses the **Enabled** property.

To use this example, follow these steps:

1 Copy this sample code to the declarations portion of an HTML Layout.

2 Add one large **TextBox** named TextBox1 to the HTML Layout.

3 Add three **Label** objects named Label1, Label2, and Label3 to the HTML Layout in a column under TextBox1.

4 Add three **TextBox** controls named TextBox2, TextBox3, and TextBox4 in a column to the right of Label1, Label2, and Label3.

```
Private Sub ShowInsPoint()
   TextBox2.Text = TextBox1.CurLine
   TextBox3.Text = TextBox1.CurX
   TextBox4.Text = TextBox1.CurTargetX
End Sub


Private Sub TextBox1_Enter()
   ShowInsPoint
End Sub


Private Sub TextBox1_KeyDown(KeyCode As Integer, ByVal Shift As Integer)
   ShowInsPoint
End Sub


Private Sub TextBox1_KeyUp(KeyCode As Integer, ByVal Shift As Integer)
   ShowInsPoint
End Sub


Private Sub UserLayout_Initialize()
   TextBox1.MultiLine = True

   TextBox1.Text = "CurTargetX identifies where, if possible, to place the
insertion point on a line within a multiline TextBox or ComboBox. The
target position is relative to the left edge of the control. If the length
of a line is less than the value of CurTargetX, you can place the insertion
point at the end of the line." _
              & Chr$(10) & Chr$(13) & "The value of CurTargetX changes when
the user sets the insertion point or when CurX is set. TheCurTargetX
property is read-only." _
              & Chr$(10) & Chr$(13) & "The position is given in himetric
units. A himetric is 0.00001 of a meter." _
              & Chr$(10) & Chr$(13) _
              & Chr$(10) & Chr$(13) & "The return value is valid when the
object has focus."

   Label1.Caption = "Current Line"
   Label1.AutoSize = True

   Label2.Caption = "Distance from left margin"
   Label2.AutoSize = True
   Label3.Caption = "Preferred distance from margin"
   Label3.AutoSize = True

   TextBox2.Enabled = False
   TextBox3.Enabled = False
   TextBox4.Enabled = False

   TextBox1.SetFocus

   TextBox1.CurLine = 0
   TextBox1.CurX = 317               'Current position expressed in
himetric units.

   ShowInsPoint
End Sub
```

accelerator key

A single character used as a shortcut for selecting an object. Pressing the ALT key followed by the accelerator key gives focus to the object and initiates one or more events associated with the object. The specific event or events initiated varies from one object to another. If code is associated with an event, it will be processed when the event is initiated. Also called keyboard accelerator, shortcut key, keyboard shortcut.

background color

The color of the client region of an empty window or display screen, on which all drawing and color display takes place.

class

The formal definition of an object. The class acts as the template from which an instance of an object is created at run time. The class defines the properties of the object and the methods used to control the object's behavior.

collection

An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

COM object

An object that conforms to the Component Object Model. COM defines how ActiveX objects and their clients interact within processes or across process boundaries.

container

An object that can contain other objects.

control

A tool you select from the ActiveX Control Pad toolbox to draw an object, such as a **CommandButton** or a **TextBox**, in an HTML Layout.

Controls have their own set of recognized properties and events. You use controls to receive user input, display output, and trigger event procedures. You can manipulate most controls using methods.

design time

The time during which you can build or modify an application in the development environment by adding controls, setting control properties, and so on. For example, during design time you can edit an .alx file in HTML Layout.

In contrast, during run time you can interact with an application as a user would.

drop source

The selected text or object that is dragged in a drag-and-drop operation.

focus

The ability to receive mouse clicks or keyboard input at any one time. In Microsoft Windows, only one window, HTML Layout, or control can have this ability at a time. The object that "has the focus" is usually indicated by a highlighted caption or title bar. The focus can be set by the user or by the application.

foreground color

The color that is currently selected for drawing or displaying text on screen. In monochrome displays, the foreground color is the color of a bitmap or other graphic.

HTML

Hypertext Markup Language. A system of marking up, or tagging, a document so it can be published on the World Wide Web. Documents prepared in HTML include reference graphics and formatting tags. You use a Web browser (such as Microsoft Internet Explorer) to view these documents.

named arguments

An argument that has a name that is predefined in the object library. Instead of providing a value for each argument in a specified order expected by the syntax, you can use named arguments to assign values in any order. For example, suppose a method accepts three arguments:

DoSomeThing *namedarg1*, *namedarg2*, *namedarg3*

By assigning values to named arguments, you can use the following statement:

```
DoSomeThing namedarg3 := 4, namedarg2 := 5, namedarg1 := 20
```

Note that the arguments don't need to appear in their normal positional order.

point

In typography, a point is 1/72 inch. The size of a font is usually expressed in points.

RGB

A color value system used to describe colors as a mixture of red (R), green (G), and blue (B). The color is defined as a set of three integers (R,G,B) where each integer ranges from 0–255. A value of 0 indicates a total absence of a color component. A value of 255 indicates the highest intensity of a color component.

run time

The time during which an application is running and you can interact with it as a user would. For example, during run time you can view an .alx file in a browser such as Internet Explorer.

In contrast, during design time you can create an application and modify its design.

single-precision value

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. The type-declaration character for Single is !.

system colors

Colors defined by the operating system for a specific type of monitor and video adapter. Each color is associated with a specific part of the user interface, such as a window title or a menu.

tab order

The order in which the focus moves from one field or object to the next as you press TAB or SHIFT+TAB.

target

An object onto which the user drops the object being dragged.

transparent

Describes the background of the object if the background is not visible. Instead of the background, you see whatever is behind the object, such as an image or picture used as a backdrop in your application. Use the **BackStyle** property to make the background transparent.

UNC

Uniform Naming Convention. The UNC specifies a directory on a server on a local area network.

The basic format is:

```
\\servername\sharename
```

where "servername" is the host name of a network file server, and "sharename" is the name of a networked or shared directory.

URL

Uniform Resource Locator. Identifies the full path of a document, graphic, or other file on the Internet or on an intranet.

A URL expresses the protocol (such as FTP or HTTP) to be accessed and the file's location. A URL may also specify an Internet e-mail address or a newsgroup. Some examples of URLs are:

http://www.someones.homepage/default.html

ftp://ftp.server.somewhere/ftp.file

gopher://server.name

file://Server/Share/File.doc

z-order

The visual layering of controls on an HTML Layout along the z-axis (depth). The z-order determines which controls are in front of other controls.

Topic Not Related
There are no topics associated with this jump.